

Teoria da Computação

Aula Teórica 19:
Linguagens LL(1)

António Ravara

Departamento de Informática

22 de Maio de 2019

Um exemplo elaborado: duas derivações de $aabbccdd$

$$\{a^n b^n c^m d^m \mid n \in \mathbb{N} \wedge m \in \mathbb{N}\} \cup \{a^n b^m c^m d^n \mid n \in \mathbb{N} \wedge m \in \mathbb{N}\}$$

► $V = \{S, A, B, C, D\}$

► $T = \{a, b, c, d\}$



$$\begin{aligned} P = \{ & S \rightarrow AB \mid C \\ & A \rightarrow aAb \mid ab \\ & B \rightarrow cBd \mid cd \\ & C \rightarrow aCd \mid aDd \\ & D \rightarrow bDc \mid bc \} \end{aligned}$$

$$S \Rightarrow_{lm} AB \Rightarrow_{lm} aAbB \Rightarrow_{lm} aabbB \Rightarrow_{lm} aabbcbBd \Rightarrow_{lm} aabbccdd$$

$$S \Rightarrow_{lm} C \Rightarrow_{lm} aCd \Rightarrow_{lm} aaDdd \Rightarrow_{lm} aabDcdd \Rightarrow_{lm} aabbccdd$$

Derivações *leftmost* e *rightmost*

- ▶ A gramática anterior pode ser simplificada (definida com menos variáveis e menos produções).
- ▶ No entanto, algumas palavras terão sempre mais que uma derivação *leftmost*.
- ▶ A gramática anterior é *ambígua*.

Ambiguidade

Uma gramática diz-se *ambígua*, se uma dada palavra da linguagem tem mais que uma derivação.

- ▶ Algumas gramáticas ambíguas têm gramáticas equivalentes não ambíguas.
- ▶ No entanto, algumas gramáticas são *inerentemente ambíguas*.
- ▶ A gramática anterior é *inerentemente ambígua*.

Remoção de ambiguidade numa CFG

- ▶ Idealmente, quer-se um algoritmo que remova a ambiguidade de uma gramática (a converta numa equivalente não ambígua).
- ▶ Surpreendentemente, tal não é possível!
- ▶ A gramática anterior é *ambígua*.

Resultados

- ▶ Há linguagens independentes de contexto que só são geradas por CFGs ambíguas – são inerentemente ambíguas.
- ▶ O problema de determinar se uma gramática é ambígua é *indecidível*. Na verdade, é *semi-decidível*: há procedimentos que respondem *não* quando uma gramática não é ambígua.

As gramáticas ambíguas são em geral mais compactas e legíveis que as não ambíguas.

CFGs para linguagens de programação

- ▶ Há gramáticas de referência de linguagens que são ambíguas.
- ▶ Resolve-se com regras de *precedência* (e/ou parêntesis).

Dangling-else problem

- ▶ Em muitas linguagens de programação o `else` dum comando condicional `if-then-else` é opcional.
- ▶ A imbricação de condicionais pode gerar palavras ambíguas.
- ▶ Considerem-se as seguintes produções de uma gramática:

$$\begin{aligned} \text{com} \rightarrow & \text{if bexpr then com} \mid \\ & \text{if bexpr then com else com} \end{aligned}$$

É possível derivar a palavra `if  $b_1$  then if  $b_2$  then  $s_1$  else  $s_2$` que pode ser interpretada como `if  $b_1$  then (if  $b_2$  then  $s_1$ ) else  $s_2$` ou como `if  $b_1$  then (if  $b_2$  then  $s_1$  else  $s_2$ )`.

CFGs Deterministas

CFG Determinista

Cada palavra da sua linguagem tem uma única derivação *leftmost*.

- ▶ CFGs não ambíguas são sempre deterministas.
- ▶ Há no entanto CFGs deterministas e ambíguas.
- ▶ As linguagens de programação devem ser CFGs deterministas e “tratáveis de forma eficiente”.

Gramáticas LL

Família de CFGs deterministas, que permitem analisar sintaticamente um *input*:

- ▶ em alguns casos, de forma linear;
- ▶ da esquerda para a direita;
- ▶ com uma derivação *leftmost*.

Toda a CFG determinista tem uma gramática LL equivalente.

Gramáticas LL(1) – são não recursivas

- ▶ A definição de gramática LL(1) requer algumas noções auxiliares.
- ▶ Estas noções são também relevantes para definir um *parser* LL(1).

Gramáticas recursivas

Uma gramática $G = \langle V, T, P, S \rangle$ diz-se *recursiva* à esquerda (resp. à direita), se permite uma derivação $X \Rightarrow^* Xw$ ($X \Rightarrow^* wX$), com $x \in V$ e $w \in \mathcal{L}_G$.

Uma gramática LL(1) não tem recursividade à esquerda.

Gramáticas LL(1) – conjunto *FIRST*

- ▶ O conjunto *FIRST* identifica o conjunto de símbolos terminais que iniciam palavras deriváveis a partir de uma variável.
- ▶ Considere-se uma gramática $G = \langle V, T, P, S \rangle$ com $a \in T$ e $X \rightarrow w_1 \mid \cdots \mid w_n \in P$ para $n \geq 1$.

$$FIRST \subseteq (V \cup T)^* \times \wp(T)$$

É a relação definida indutivamente pelas seguintes regras:

- ▶ $FIRST(X) = \bigcup_{X \rightarrow w_i} FIRST(w_i)$, com $1 \leq i \leq n$
- ▶ $FIRST(\epsilon) = \emptyset$
- ▶ $FIRST(a w) = \{a\}$
- ▶ $FIRST(X w) = FIRST(X) \cup \begin{cases} \emptyset & \text{se } \forall i w_i \not\Rightarrow^* \epsilon \\ FIRST(w) & \text{c.c.} \end{cases}$

Conjunto *FIRST* – exemplo

Considere-se uma gramática com as seguintes produções:

$$S \rightarrow ABc \mid BAc$$

$$A \rightarrow a \mid aA$$

$$B \rightarrow b \mid bB$$

- ▶ $FIRST(ABc) = FIRST(A) = FIRST(a) \cup FIRST(aA) = \{a\}$
- ▶ $FIRST(BAc) = FIRST(B) = \{b\}$
- ▶ $FIRST(S) = FIRST(ABc) \cup FIRST(BAc) = \{a, b\}$

Gramáticas LL(1) – conjunto *FOLLOW*

- ▶ O conjunto *FOLLOW* identifica o conjunto de símbolos terminais que surgem no início de palavras deriváveis a partir de uma variável.
- ▶ Considere-se uma gramática $G = \langle V, T, P, S \rangle$.

$$FOLLOW \subseteq V \times \wp(T)$$

É a relação definida indutivamente pela seguinte regra:

$$FOLLOW(X) = \bigcup_{(Y \rightarrow w_1 X w_2) \in P} FIRST(w_2) \cup \begin{cases} \emptyset & \text{se } w_2 \not\stackrel{*}{\Rightarrow} \epsilon \\ FOLLOW(Y) & \text{c.c.} \end{cases}$$

Conjunto *FOLLOW* – exemplo

Considere-se de novo a gramática com as seguintes produções:

$$S \rightarrow ABc \mid BA c$$

$$A \rightarrow a \mid aA$$

$$B \rightarrow b \mid bB$$

com $FIRST(S) = \{a, b\}$, $FIRST(A) = \{a\}$ e $FIRST(B) = \{b\}$.

- ▶ $FOLLOW(S) = \emptyset$, pois S não surge no lado direito de nenhuma produção.
- ▶ Como A surge no lado direito de 3 produções, calcula-se para cada uma e unem-se os resultados:

$$FOLLOW(A) = FIRST(Bc) \cup FIRST(c) \cup FIRST(\epsilon) \cup FOLLOW(A)$$

$$FOLLOW(A) = FIRST(B) \cup \{c\} \cup \emptyset \cup FOLLOW(A)$$

$$FOLLOW(A) = \epsilon^* \{b, c\} = \{b, c\}, \text{ pelo Lema de Arden.}$$

Gramáticas LL(1) – definição

Considere-se uma gramática $G = \langle V, T, P, S \rangle$ e $X \rightarrow w \in P$.

$$\text{Lookahead} \subseteq V \times (V \cup T)^* \times \wp(T)$$

$$\text{Lookahead}(X \rightarrow w) = \text{FIRST}(w) \cup \begin{cases} \emptyset & \text{se } w \not\stackrel{*}{\Rightarrow} \epsilon \\ \text{FOLLOW}(X) & \text{c.c.} \end{cases}$$

Definição

Uma CFG G é uma gramática LL(1) se

1. não apresenta recursividade à esquerda;
evita derivações infinitas;
2. se $X \rightarrow w_1 \in P$ e $X \rightarrow w_2 \in P$ então
 $\text{Lookahead}(X \rightarrow w_1) \cap \text{Lookahead}(X \rightarrow w_2) = \emptyset$
é determinista
(só tem uma regra para aplicar em cada passo de uma derivação).

Gramáticas LL(1) – exemplo

Exemplo

A gramática com as produções

$$S \rightarrow ABc \mid BAc$$

$$A \rightarrow a \mid aA$$

$$B \rightarrow b \mid bB$$

é LL(1)?

1. não apresenta recursividade à esquerda;
2. tem conflitos?

$$\begin{aligned} 2.1 \quad & \text{Lookahead}(S \rightarrow ABc) = \text{FIRST}(ABc) = \{a\}; \\ & \text{Lookahead}(S \rightarrow BAc) = \text{FIRST}(BAc) = \{b\}; \end{aligned}$$

$$\begin{aligned} 2.2 \quad & \text{Lookahead}(A \rightarrow a) = \text{FIRST}(a) = \{a\}; \\ & \text{Lookahead}(A \rightarrow aA) = \text{FIRST}(aA) = \{a\}; \end{aligned}$$

sim – não é LL(1).

Uma gramática equivalente á anterior

A gramática com as produções seguintes é LL(1):

Não apresenta recursividade à esquerda nem tem conflitos.

$$S \rightarrow ABc \mid BAc$$

$$A \rightarrow aD$$

$$D \rightarrow \epsilon \mid aD$$

$$B \rightarrow bE$$

$$E \rightarrow \epsilon \mid bE$$

$$\text{Lookahead}(S \rightarrow ABc) \neq \text{Lookahead}(S \rightarrow BAc)$$

$$\begin{aligned}\text{Lookahead}(D \rightarrow \epsilon) &= \text{FIRST}(\epsilon) \cup \text{FOLLOW}(D) \\ &= \emptyset \cup \text{FOLLOW}(A) \\ &= \text{FIRST}(Bc) \cup \text{FIRST}(c) \\ &= \text{FIRST}(B) \cup \{c\} = \text{FIRST}(bE) \cup \{c\} \\ &= \{b, c\} \neq \{a\} = \text{Lookahead}(D \rightarrow aD)\end{aligned}$$